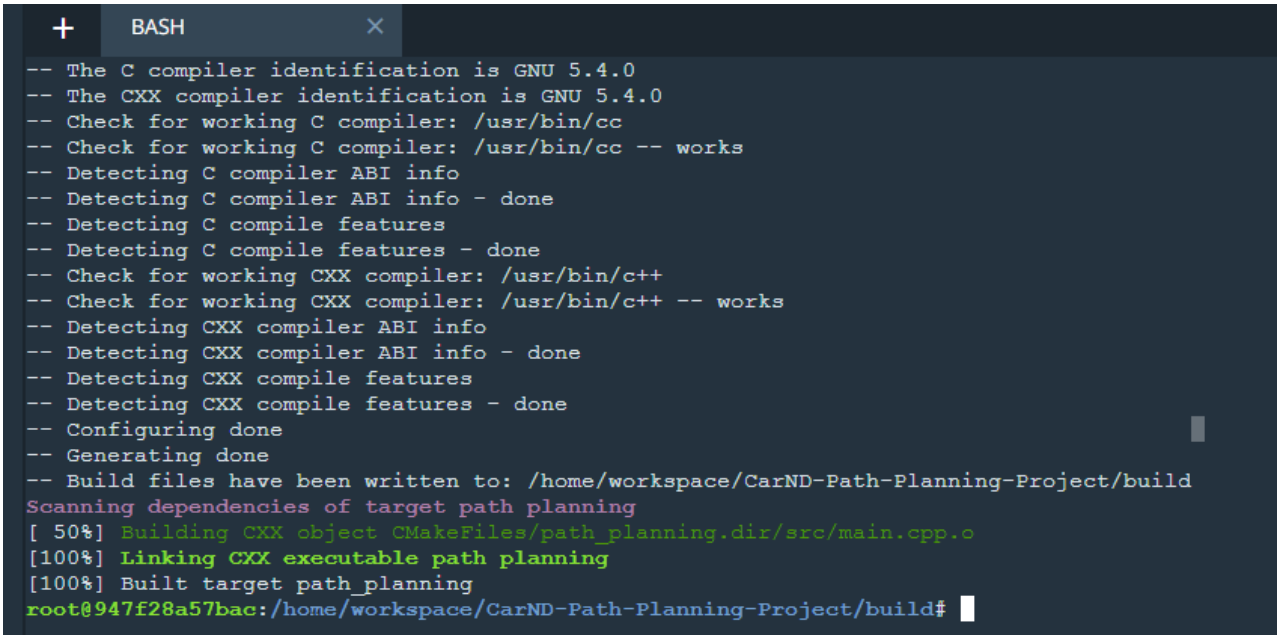
Path Planning Project Writeup

Goals

The goal of this project is to make the car safely navigate a highway with other vehicles that is driving 50Mph +- 10Mph. The car should pass slower traffic when possible and avoid hitting other cars at all cost. Also the car must stay within the lanes and make one loop of the highway while successfully changing lanes . Acceleration must stay below 10m/s^2 and jerk less than 10m/s^3

Compilation

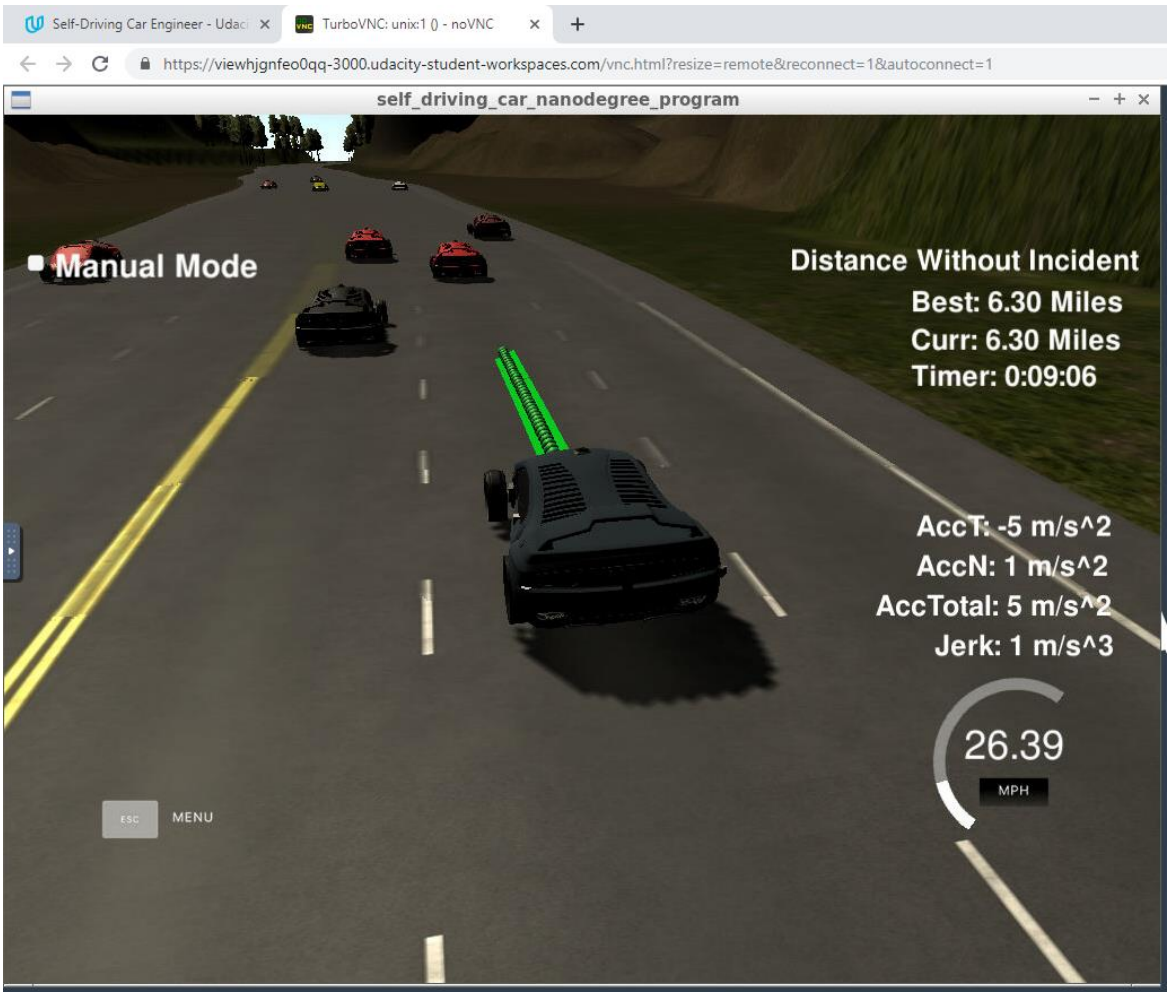
Code compiles without errors with cmake and make.



Valid Trajectories

1.The car can drive at least 4.32 miles without incident.

The car ran 6.3 miles without incidents



2.The car drives according to the speed limit.

The car drives within the speed limits all the time.

3.Max Acceleration and Jerk are not Exceeded.

The car does not exceed a total acceleration of 10 m/s^2 and a jerk of 10 m/s^3. Max jerk exceeded message was not seen.

4.Car does not have collisions.

The car does not collide with any of the other vehicles on the road. Collisions occurred message was not seen.

5.The car stays in its lane, except for the time between changing lanes.

Most of the time, the car stays in its lane, but change lanes when ahead car speed is slow or change to center lane.

6.The car is able to change lanes

The car change lanes when ahead car speed is slow or change to center lane.

Reflection

1.Prediction no-ego cars

Use the sensor fusion data to find whether there has a no-go car ahead from go car, whether there has a no-go car on the left/right lane will affect the go car to change lane. Calculating the lane each no-go car is and the position it will be when at the end of the previous plan trajectory. When the distance between no-go car and go car is less than 30 m, it is considered as not safe.

```
260                 // prediction, detect other car's positions
261                 bool car_go_ahead = false;
262                 bool car_on_left = false;
263                 bool car_on_right = false;
264
265
266                 for(int i =0;i<sensor_fusion.size();i++)
267                 {
268                     float d = sensor_fusion[i][6];
269                     int car_lane =-1;
270                     //first check which lane the no-go vehicle is
271                     if(d>=0 && d<4)
272                     {
273                         car_lane = 0;
274                     }
275                     else if(d>=4 && d<8)
276                     {
277                         car_lane = 1;
278                     }
279                     else if(d>=8 && d<=12)
280                     {
281                         car_lane =2;
282                     }
283                     else
284                     {
285                         //do nothing
286                     }
287
288                     if(car_lane <0)
289                     {
290                         continue;
291                     }
```

```
292
293                     // car speed
294                     double vx =sensor_fusion[i][3];
295                     double vy =sensor_fusion[i][4];
296
297                     double check_speed =sqrt(vx*vx+vy*vy);
298                     double check_car_s = sensor_fusion[i][5];
299
300                     // if using previous points, can project s value outwards in time
301                     check_car_s += ((double)prev_size*0.02*check_speed);
302
303
304                     if (car_lane == lane)
305                     {
306                         //no-go car in our lane
307                         car_go_ahead |= check_car_s > car_s && check_car_s - car_s <30;
308                     }
309                     else if (car_lane -lane == -1)
310                     {
311                         //no-go car left
312                         car_on_left |= car_s-check_car_s<30 && check_car_s-car_s<30;
313                     }
314                     else if (car_lane- lane == 1)
315                     {
316                         //no-go car right
317                         car_on_right |= car_s-check_car_s<30 && check_car_s-car_s<30;
318                     }
319                     else{}
320
321
```

2.Behavior planner

Given the prediction of the no-go cars around go car, when a slow no-go car ahead ego car, left and right direction do have no lane to change or it is not safe to change lane, the no-go car will slow down. If a slow no-go car ahead go car, and left or right lane is safe to change, the no-go will change the lane. If no-go car is not in center lane and it is safe to change to center lane, the no-go car will get back to center lane. Use speed_diff to create the speed change, when generating trajectory use this change will make the car more responsive to act the speed change.

```
                //behavior planner
                double speed_diff = 0;
                const double MAX_SPEED =49.5;
                const double MAX_ACC =0.224;
                if(car_go_ahead)
                {
                    //no-go car ahead
                    if (!car_on_left && lane > 0)
                    {
                        //if there is no other car left and there is a left lane.
                        lane--; // for changing to the left lane
                    }
                    else if(!car_on_right && lane !=2)
                    {
                        // if there is no other car right and there is a right lane.
                        lane++; //for changing to the right lane
                    }
                    else
                    {
                        speed_diff -= MAX_ACC;
                    }
                }
                else
                {
                    if(lane!=1)
                    {
                        // if car isnt' on the center lane.
                        if ((lane ==0 && !car_on_right) || (lane == 2 && !car_on_left))
                        {
                            lane =1; // for going back to the center
                        }
                    }

                  if (ref_vel < MAX_SPEED)
                  {
                      speed_diff += MAX_ACC;
                  }
              }


```

3.Trajectory generation

The last two points of the previous trajectory or the car position (when there is almost no previous trajectory) are used to combine with other three far distance points to initialize the spline. To make live easy, the coordinates are transformed to local car coordinates from map coordinate. To make the trajectory smoother, the rest previous trajectory points are used with the new trajectory. The new trajectory is calculating by evaluating the spline, then all the trajectory changes the coordinates from local car coordinates to map coordinate. The speed value will affect the space between two points the spline evaluated.

```
367                 vector<double> ptsx;
368                 vector<double> ptsy;
369
370                 //reference x,y,yaw sates, either starting point where the car is or the previous paths end point
371
372                 double ref_x = car_x;
373                 double ref_y = car_y;
374                 double ref_yaw = deg2rad(car_yaw);
375
376                 //if previous size is almost empty, use the car as starting reference
377                 if(prev_size < 2)
378                 {
379                     //use two points that make the path tangent to car
380                     double prev_car_x = car_x - cos(car_yaw);
381                     double prev_car_y = car_y - sin(car_yaw);
382
383                     ptsx.push_back(prev_car_x);
384                     ptsx.push_back(car_x);
385
386                     ptsy.push_back(prev_car_y);
387                     ptsy.push_back(car_y);
388                 }
389                 //use the previous path's end point as starting reference
390                 else
391                 {
392                     //redefine reference state as previous path end point
393                     ref_x = previous_path_x[prev_size-1];
394                     ref_y = previous_path_y[prev_size-1];
395
396                     double ref_x_prev = previous_path_x[prev_size-2];
397                     double ref_y_prev = previous_path_y[prev_size-2];
398                     ref_yaw = atan2(ref_y - ref_y_prev, ref_x - ref_x_prev);
```

```
                    // use two points that make the path tangent to the previous path's end point
                    ptsx.push_back(ref_x_prev);
                    ptsx.push_back(ref_x);

                    ptsy.push_back(ref_y_prev);
                    ptsy.push_back(ref_y);
                }

                // in Frenet space adding 30m spaced points ahead of the starting reference
                vector<double> next_wp0 = getXY(car_s+30, (2+ 4*lane),map_waypoints_s,map_waypoints_x,map_waypoints_y);
                vector<double> next_wp1 = getXY(car_s+60, (2+ 4*lane),map_waypoints_s,map_waypoints_x,map_waypoints_y);
                vector<double> next_wp2 = getXY(car_s+90, (2+ 4*lane),map_waypoints_s,map_waypoints_x,map_waypoints_y);

                ptsx.push_back(next_wp0[0]);
                ptsx.push_back(next_wp1[0]);
                ptsx.push_back(next_wp2[0]);

                ptsy.push_back(next_wp0[1]);
                ptsy.push_back(next_wp1[1]);
                ptsy.push_back(next_wp2[1]);

                for(int i = 0; i< ptsx.size(); i++)
                {
                  // car reference angle reverted back to 0 degress
                  double shift_x = ptsx[i]-ref_x;
                  double shift_y = ptsy[i]-ref_y;

                  ptsx[i] = shift_x *cos(0-ref_yaw) - shift_y*sin(0-ref_yaw);
                  ptsy[i] = shift_x *sin(0-ref_yaw) + shift_y*cos(0-ref_yaw);

                }

```

```
                // spline
                tk::spline s;

                // set (x,y) points to the spline
                s.set_points(ptsx,ptsy);

                //define the acutal (x,y) points we will use for the planner
            vector<double> next_x_vals;
            vector<double> next_y_vals;

                //start with all of the previous path points from last time
                for(int i = 0; i<previous_path_x.size();i++)
                {
                   next_x_vals.push_back(previous_path_x[i]);
                   next_y_vals.push_back(previous_path_y[i]);
                }

                //calculate how to break up spline points so that  we travel at our desired reference velocity
                double target_x = 30.0;
                double target_y = s(target_x);
                double target_dist = sqrt(target_x*target_x + target_y*target_y);

                double x_add_on = 0;

                //fill up the rest of our path planner after filling it with previous points, here we will always output 50 points
                for(int i =1; i <= 50- previous_path_x.size(); i++)
                {

                    ref_vel += speed_diff;
                    if (ref_vel >MAX_SPEED)
                    {
                        ref_vel = MAX_SPEED;
```

```
465                     else if (ref_vel < MAX_ACC)
466                     {
467                         ref_vel = MAX_ACC;
468                     }
469                     else
470                     {
471                         //do nothing
472                     }
473 
474                     double N = (target_dist/(0.02*ref_vel/2.24));
475                     double x_point = x_add_on + target_x/N;
476                     double y_point = s(x_point);
477 
478                     x_add_on = x_point;
479 
480                     double x_ref = x_point;
481                     double y_ref = y_point;
482 
483                     //rotate back to normal after rotating it eariler
484                     x_point = x_ref * cos(ref_yaw) - y_ref*sin(ref_yaw);
485                     y_point = x_ref * sin(ref_yaw) + y_ref*cos(ref_yaw);
486 
487                     x_point += ref_x;
488                     y_point += ref_y;
489 
490                     next_x_vals.push_back(x_point);
491                     next_y_vals.push_back(y_point);
492 
493                 }
```